CHAPTER 4

Ensuring Application Availability

Application availability is the readiness of an application and the service it runs under to handle customer requests and to return timely and accurate responses. Ensuring consistently high application availability requires that you set availability goals that meet your business needs, configure your Web server to achieve those goals, and measure the compatibility and reliability of your applications. Internet Information Services (IIS) 6.0 running in worker process isolation mode provides the advanced features that you need to achieve and maintain a high degree of application availability.

In This Chapter

Related Information

- For information about migrating IIS applications, see “Migrating IIS Web Sites to IIS 6.0” in this book.
- For information about securing IIS applications, see “Securing Web Sites and Applications” in this book.
- For information about migrating Apache Web sites, see “Migrating Apache Web Sites to IIS 6.0” in this book.

Overview of the Ensuring Application Availability Process

To help ensure optimum availability of your applications to users who need them, you need to do more than ensure that the network is available and that your server is up and running. You need to set availability goals, configure IIS to meet the demands that users are placing on your applications, and test your applications for functional compatibility with IIS 6.0 worker process isolation mode.

To be certain that the level of application availability on your server meets the needs of your customers, you must first establish availability, service, and request-handling goals that accurately represent those customer needs. Next, you need to create application pools, and then configure IIS features to isolate applications and to tune and monitor your application pools. Finally, you need to assess the tradeoffs between availability and other aspects of running your applications, such as performance.

To help maintain high availability of the applications that your server hosts, you can use two kinds of testing: compatibility testing and comparison testing. Functional compatibility testing verifies that your applications will work with the IIS 6.0 worker process isolation mode features that you have enabled. Comparison testing measures the availability of your applications against your application availability goals, which reveals how closely the day-to-day availability of applications hosted on your server matches your original goals.

Before you begin this process, you must:

- Install the Microsoft® Windows® Server 2003, Standard Edition; Windows® Server 2003, Enterprise Edition; Windows® Server 2003, Datacenter Edition; or Windows® Server 2003, Web Edition operating system with the default options.
- Install IIS 6.0 with the default settings by using Add or Remove Programs in Control Panel.

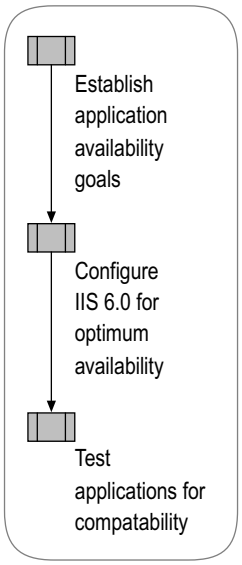
If you use other methods for installing and configuring Windows Server 2003, such as unattended setup, or enabling IIS 6.0 by using Manage Your Server, then the default configuration settings might not be identical to the configuration settings described in this chapter.

When you complete the process for ensuring application availability, you will meet the basic requirements for achieving high availability for your applications. You will set and begin to measure availability goals for you applications based on your business needs. Your applications will be configured to make effective use of IIS 6.0 features. Also, you will know whether your applications are compatible with IIS 6.0 worker process isolation mode.

Process for Ensuring Application Availability

The process for ensuring application availability includes setting availability goals, configuring IIS to achieve your availability goals, and testing your applications to ensure that they meet the availability goals that you set. Ensure that your availability goals accommodate the needs of your users for high application reliability and reasonable application response times. Figure 4.1 shows the process for ensuring application availability.

Figure 4.1 Ensuring Application Availability



Application pools are sets of applications and the worker processes that service them. For you to create and configure application pools and make application pool assignments, IIS 6.0 must be running in worker process isolation mode. Thus, you need to evaluate the compatibility of your applications with IIS 6.0 worker process isolation mode in a test environment before deploying them in your production environment.

Important

The application-pool functionality is only available in IIS 6.0 worker process isolation mode.

The following quick-start guide provides a detailed overview of how to help ensure the availability of your applications. You can use this guide to help identify the steps of the ensuring application availability process that you need additional information about to complete so that you can disregard the information with which you are already familiar. In addition, for the procedures that are required to complete this process, see “IIS Deployment Procedures in this book.

Establishing Application Availability Goals

1. Set service availability goals.
2. Set request-handling reliability goals.

Configuring IIS 6.0 for Optimum Availability

1. Isolate applications by completing the following steps:
   - Determine the application isolation needs of your server.
   - Create application pools and assign applications to them.
2. Recycle worker processes in one of the following ways:
   - Recycle by elapsed time.
   - Recycle by number of requests.
   - Recycle at scheduled times.
   - Recycle on a virtual-memory threshold.
   - Recycle on a used-memory threshold.
3. Tune performance by completing the following steps:
   - Configure idle time-out for worker processes.
   - Configure a request queue limit.
   - Configure Web gardens.
   - Set processor affinity on servers that include multiple CPUs.
4. Manage application pool health by completing the following steps:
   - Configure worker process pinging.
   - Configure rapid-fail protection for worker processes.
   - Configure the startup time limit for worker processes.
   - Configure the shutdown time limit for worker processes.
   - Enable debugging for application pool failures.
5. Configure application pool identity.

Testing Applications for Compatibility

6. Test applications for setup compatibility with IIS 6.0.
7. Test applications for functional compatibility with IIS 6.0.

Establishing Application Availability Goals

To set effective availability goals for your applications, you need to measure the availability of the World Wide Web Publishing service (WWW service), your Web sites, and the worker processes that service your applications. These availability measurements can be calculated as a percentage by dividing the uptime of the WWW service, Web site, or worker process by the time elapsed. In addition, a good user application availability experience depends upon successful handling of correctly formed requests, which are requests that are complete and error-free.

At times, users cannot access your applications because of these common request-handling problems:

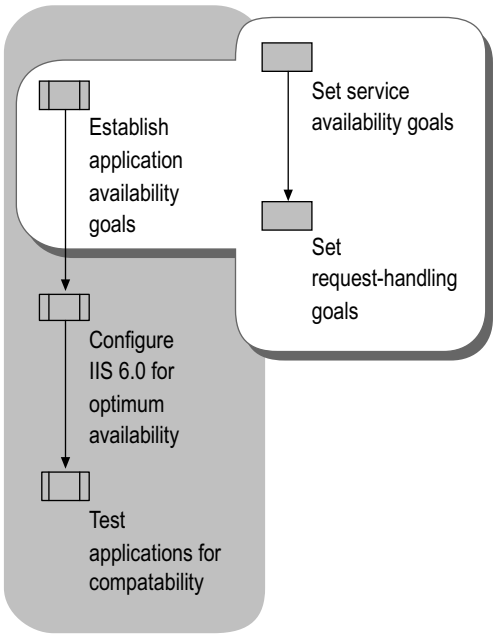
- Correctly formed requests are sent, but they are not correctly served.
- Correctly formed requests are sent, but they are served only after a time interval that is unacceptable to the user.

To ensure that you set application availability goals that represent the needs of your users, you must first define how you will measure the successful serving of requests. Make sure that your goals address the following:

- Service availability, which includes the WWW service and individual Web sites.
- Application availability, which includes request processing reliability and the average time that is needed to process requests.

Figure 4.2 shows the process for establishing application availability goals.

Figure 4.2 Establishing Application Availability Goals



Setting Service Availability Goals

To set effective service availability goals, you need to measure the percentage of time that the WWW service and your Web sites are available to process client requests. WWW service availability is the time that the WWW service and your Web sites are up and running, expressed as a percentage of a set period of time. For example, an organization might set a goal to have a WWW service availability of 99.999 percent over a 21-day period.

You can achieve high WWW service availability when running IIS 6.0 in worker process isolation mode. This is because worker process isolation mode isolates the WWW service from the impacts of failed applications. Generally, the WWW service is unavailable only when the system is restarted and when IIS is taken offline for major service upgrades, such as the installation of a service pack. When the WWW service is available, your Web sites are available unless you deliberately take them offline.

Test service availability and compare the test results to your goals. First, run the tests in a test environment, and then run the tests periodically in your production environment.

Measuring Service Availability

You can measure WWW service availability by completing the following steps:

8. Use Performance Monitor to gather the data for WWW service uptime, which measures uptime for w3svc.exe, and uptime for each Web site on the server. The data for WWW service uptime is displayed in the Total instance of the Service Uptime counter. The data for Web site uptime is displayed in the Sites instances of the same counter. For more information about gathering uptime data for the WWW service and Web sites, see "Gather and Display WWW Service Uptime Data" in "IIS Deployment Procedures" in this book.
9. Use Service Monitor to display the data for the Service Uptime counter. The Service Monitor displays the uptime data for the WWW service and for each Web site on the server. For more information about displaying WWW service and Web site uptime data, see "Gather and Display WWW Service Uptime Data" in "IIS Deployment Procedures" in this book.
10. Calculate the availability of the WWW service by dividing the time (in seconds) for the Total instance by the elapsed time that the server was running (in seconds) during the test; multiply the result by 100 to express the figure as a percent availability.
11. Calculate the availability of each Web site by dividing the time (in seconds) for the Site instance for each site by the elapsed time (in seconds) that the server was running during the test; multiply the result by 100 to express the figure as a percent availability.
12. Compare the results to your goals.

Setting Request-Handling Goals

Request handling measures whether client requests were served correctly and the time needed for the WWW service to process client requests. There are two important measures for request handling: *request-handling reliability* and *request processing time*. Request-handling reliability measures the percentage of correctly formed requests that are returned successfully; request processing time measures the average elapsed time for IIS to process a client request.

Request-Handling Reliability

For a range of request rates that your applications receive, set goals to achieve the level of overall request handling reliability that you want. Measure only correct requests. Do not consider a request that a client composed incorrectly as a failed request.

For example, you can establish a request processing reliability goal of 99.999% for a rate of 20 correctly formed requests per second. You meet this goal when a Web server that receives requests at a rate of 20 per second successfully serves 99,999 responses out of each 100,000 requests. A response may be considered successful if the request returns the expected message, such as 200 OK, and the correct content.

When testing your applications in a test environment, you can use a request rate such as 20 requests per second. However, when testing applications in a production environment, you might need to set a range of request rates. If you estimate that 20 requests per second is a reasonable goal, you might try a range of 10-30 requests per second. Also, monitor request handling over a long period of time, such as 21 days or a month, to ensure that you cover the periods of heaviest loads on the application.

The following is an example of a full expression of a request reliability goal for a Web site or application:

- Duration: 21 days
- Load: 10-30 requests per second
- Success rate goal: 99.999 percent responses with the expected return message, such as 200 OK, measured against 100,000 requests.

Keep in mind that you must adjust your goals to accommodate your needs, taking into account client expectations, application backend processing, and caching.

Request Processing Time

When a request is processed so slowly that the user is not willing to wait for a return, the effect on the user is similar that of to a request failure. The user experiences a loss of availability.

The following is an example of a request reliability goal that accommodates request-handling execution time:

- Duration: 21 days
- Load: 10-30 requests per second
- Success rate goal: 99.999 percent responses with the expected return message, such as 200 OK, measured against 100,000 requests.
- Average response time of 5 seconds for each request

To set your own goals for request processing time, evaluate the availability requirements of your organization and the equipment and software that you are using.

Testing Request-Handling

Test request handling and compare the test results to your goals. First run tests in a test environment, and then periodically run tests in your production environment. Measure request-handling performance for an application by completing the following steps:

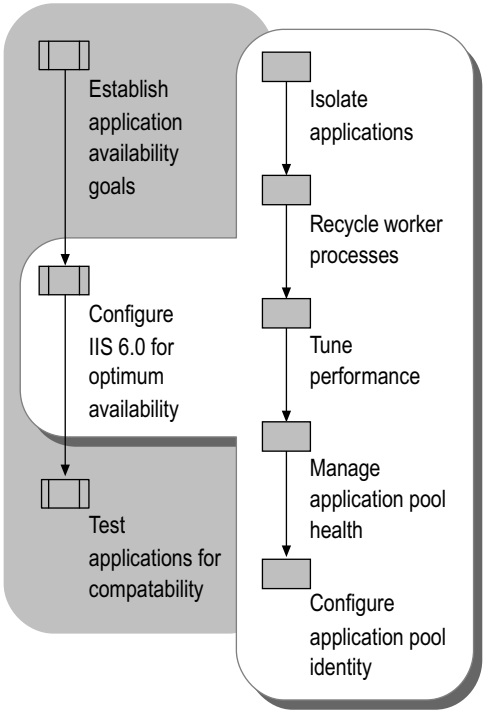
13. Send correctly formed requests from a remote client to the target application on the server that is hosting the application that you are testing. If you are testing an application on a production server, measure the performance of requests sent by users. If you are testing an application running on a test server, use automated load-testing software, such as WebCAT, to send client requests to the application at a rate appropriate to test it. You can use WebCAT to automate sending customized request loads to simulate a wide variety of conditions. You can use Log parser to create request schemes that WebCAT can use. For information about WebCAT and Log parser, see the *Internet Information Services (IIS) 6.0 Resource Guide* of the *Microsoft® Windows® Server 2003 Resource Kit* (or see the *Internet Information Services (IIS) 6.0 Resource Guide* on the Web at http://www.microsoft.com/reskit).
14. Measure the rate at which the correctly formed requests are being processed successfully by the target application on your test server (responses that return 200 messages and correct content).
15. Calculate the average elapsed time to process requests. One way to do this is to average the readings of the TimeTaken counter. The readings are in the IIS log.
16. Compare the results to your goals.

Configuring IIS 6.0 for Optimum Availability

Worker process isolation mode allows you to optimize the availability of applications installed on a Web server. To configure worker process isolation mode for optimum availability of your IIS 6.0 applications and Web sites, you need to determine availability requirements for each application and isolate applications into application pools based on those needs. You can then configure each application pool for recycling, health monitoring, performance, and identity to meet your availability requirements.

Figure 4.3 shows the process for configuring IIS for optimum application availability.

Figure 4.3 Configuring IIS 6.0 for Optimum Availability



Isolating Applications

Because applications in an application pool are isolated from the applications in other application pools by worker process boundaries, if an application fails, it does not affect the availability of applications that are running in other application pools. Deploying applications in application pools is a primary advantage of running IIS 6.0 in worker process isolation mode because you can customize the application pools to achieve the degree of application isolation that you need.

Configuring application isolation involves completing the following steps:

- Categorizing applications according to their isolation profiles for initial deployment.
- Creating and naming the application pools.

- Assigning applications to the appropriate application pools.

Note

When you configure application pools for optimum availability, also consider how to configure application pools for application security. For example, you might need to create separate application pools for applications that require a high level of security, while allowing applications that require a lower level of security to share the same application pool. Also, client requests might be running identities that are different from the identities that are assigned to the applications that serve them, which can make the application unavailable to customers. For more information about securing applications by configuring applications pools, see “Securing Web Sites and Applications” in this book.

If your organization includes network configurations that require more than one server for implementation, such as Web farms, this involves clustering and load balancing. For more information about clustering and load balancing and the specific processes for setting up replication and clustering on IIS, search for “clustering” and “load balancing” in Help and Support Center for Windows Server 2003.

Determining the Application Isolation Needs of Your Server

The applications that you deploy on a server might fit into several categories, based on your assessment of their importance, demand for server resources, or history of instability. Group your applications according to the following categories, or make up your own set of application categories, before assigning your applications to application pools:

- **Typical.** The application is stable and is not subject to extremely high user demand.
- **High-demand.** The application is expected to receive a large number of requests, putting stress on your the resources of your server.
- **Mission-critical.** The application availability is mandatory for business purposes.
- **Problem.** The application must be isolated from other applications and Web sites due to known or anticipated unstable behavior. If you must run test versions of an application on the same server that runs production applications and Web sites, consider it a problem application that must be isolated from all of the other applications on the server.
- **Unique.** The application requires a configuration that is different from other applications that you have categorized as typical. For example, an application might require an identity that has access to more system resources than the default identity.

Creating Application Pools and Assigning Applications to Them

After establishing similar groups for your applications and Web sites, you can assign them to application pools according to the degree of application isolation they require.

To create application pools and assign applications to them when you are deploying a set of Web sites or applications of different types:

17. Create one application pool for all of the applications that you categorized as typical.
18. Assign all of the applications categorized as typical to the application pool that you created in Step 1.
19. Create one application pool for each high-demand, mission-critical, problem, and unique application. Name each application pool after the application that it will contain.
20. Assign each of your high-demand, mission-critical, problem, and unique applications to the corresponding application pool that you created and named in Step 3.

    You can add application pools by using IIS Manager, by using a command-line procedure, or by running Active Directory® Service Interfaces (ADSI)– or Windows Management Instrumentation (WMI)–compatible scripts. It is recommended that you use IIS Manager because IIS Manager provides clear, complete, and well-organized visibility to application pools and their properties.

For more information about how to create and name application pools, and how to assign applications to application pools through IIS Manager, see "Isolate Applications in Worker Process Isolation Mode" in "IIS Deployment Procedures" in this book. For information about how to assign applications to application pools by using the command line or scripts, see the **AppPoolId** property in the "Metabase Property Reference" in IIS 6.0 Help, which is accessible from IIS Manager.

If you create a large number of application pools running many worker processes concurrently, server performance may suffer. The key factor to consider is that large numbers of worker processes with dynamic applications (ASP or ASP.NET applications) may use up available memory.

Unless you have specific reasons for altering the settings for an application pool, deploy your application pools using the default settings. For more information about specific reasons for using non-default settings, see "Recycling Worker Processes," "Tuning Performance," "Managing Application Pool Health," and "Configuring Application Pool Identity" later in this chapter.

Recycling Worker Processes

If an application contains code that causes problems, and you cannot easily rewrite the code, it might be useful to limit the extent of the problems by periodically recycling the worker process that services the application. *Worker process recycling* is the replacing of the instance of the application in memory. IIS 6.0 can automatically recycle worker processes by restarting the worker process, or worker processes, that are assigned to an application pool. This helps keep problematic applications running smoothly, and minimizes problems such as memory leaks.

You can trigger the recycling of the worker processes assigned to an application pool by using worker process recycling methods that are based on elapsed time, the number of Hypertext Transfer Protocol (HTTP) requests, a set time of day, and two kinds of memory consumption, in addition to recycling on demand. Table 4.1 lists and describes the worker process recycling methods and when to use each.

Table 4.1 Worker Process Recycling Methods

Method	Description	Usage
Elapsed time	**Recycles the worker process based on a time frame, in minutes, that is specified by the administrator.**	**Use this method when applications have problems running for extended time periods.**
Number of requests	**Recycles the worker process when the number of HTTP requests exceeds a threshold that is specified by the administrator.**	**Use this method when applications are failing based on the number of requests they receive.**
Scheduled times	**Recycles the worker process at specified times within a 24-hour period.**	**Use this method when conditions are similar to the conditions for elapsed time.**
Virtual memory	**Recycles the worker process when the worker process virtual memory reaches a threshold that is specified by the administrator.**	**Use this method when the memory heap becomes highly fragmented, which occurs when applications reserve memory multiple times. This is indicated by a steady increase in virtual memory.**
Used memory	**Recycles the worker process when the memory that is used by the W3wp.exe process reaches a threshold that is specified by the administrator.**	**Use this method when some of your applications have memory leaks.**
On demand	**Recycles the worker process when an administrator uses Microsoft Management Console (MMC) or a script to request the recycling of an entire application pool.**	**Use this method when an application pool is causing problems while the other sites are up and running. Consider recycling the application pool, as opposed to resetting the entire WWW service.**

By setting the values for worker process recycling methods, you can schedule recycling events to manage application problems in a timely manner. Table 4.2 lists recommendations for adjusting the values of worker process recycling methods.

Table 4.2 Recommended Settings for Values of Worker Process Recycling Methods

Recycling Method	Recommended Setting
Elapsed time	Set this value to be less than the length of time elapsed before the worker process fails.
Number of requests	Set this value to be less than the number of requests processed before the application fails.
Scheduled times	Set this value to be at intervals that are frequent enough to prevent application failure.
Virtual memory	Initially, set this value to be less than 70 percent of available virtual memory. Then monitor the memory usage of the worker process, and adjust the setting if memory usage continues to increase. Calculate this value in megabytes.
Used memory	Set this value to be less than 60 percent of available physical memory. Calculate this value in megabytes.
On demand	Consider using one of the automated methods for recycling instead, because on-demand recycling is typically performed in response to a failure.

Recycling and Maintaining State

If you have an application pool with applications that depend on state data, then you must decide whether or not to recycle the worker processes that are assigned to that application pool. When you recycle a worker process, the state data for applications that is maintained in process is lost. In this case, do not use recycling.

One alternative to increasing application availability by recycling worker processes is to maintain state data external to the worker process, such as in a database. However, moving data to an external database to allow recycling can affect server performance in the following two ways:

- Performance is reduced because of the added data management that is needed to move the data between the application and the database.
- Recycling flushes any in-process data caches, so the caches need to be rebuilt.

ASP.NET gives you the option of persisting session state using a session state service or a SQL database. For more information on deploying ASP.NET applications, see “Deploying ASP.NET Applications in IIS 6.0” in this book.

Global Locks and Overlapped Recycling

Using global locks, configuring files for exclusive read, and other methods of exclusive resource ownership prevent the use of *overlapped recycling*, or the loading of an application in two application pools simultaneously. Exlcusive ownership requires the use of non-overlapped recycling and the routing of all requests for that application to the same application pool. Also, you can run only one worker process for that application pool, which means that exclusive ownership does not work in a Web garden. One way to work around this is to write multithreaded applications, or use shared reads, which might eliminate the need for exclusively owned resources.

Recycling by Elapsed Time

You can configure an application pool to recycle the worker process, or worker processes, that are assigned to it at specified intervals. By default, recycling by elapsed time is enabled in IIS 6.0, with the elapsed time set to 29 hours. This default setting fits the recycling needs of applications that are problem-free, or that have flaws that cause problems gradually. Recycling worker processes at this interval refreshes applications often enough to correct problems that build up slowly, without creating the severe overhead that can be caused by closely spaced recycling.

Reset the elapsed-time interval for recycling a worker process if your tests or log records indicate that the application that the worker process is assigned to is encountering problems after a given period of uptime. Set the elapsed time to a shorter period of time if problems are building up in less than the default time interval of 29 hours.

Microsoft does not recommend recycling ASP.NET applications by elapsed time.

For information about how to use IIS Manager to configure a worker process to be recycled after a set elapsed time, see “Configure Application Pool Recycling” in “IIS Deployment Procedures” in this book. For information about how to configure a worker process to be recycled after a set elapsed time by using a command-line procedure or scripts, see the **PeriodicRestartTime** property in the “Metabase Property Reference” in IIS 6.0 Help.

Recycling by Number of Requests

You can configure an application pool to recycle the worker process, or processes, that are assigned to it whenever the worker process handles a specified number of requests since the last recycle. By default, recycling by number of requests is disabled in IIS 6.0. When this feature is enabled, the default is 35,000 requests.

Enable recycling by number of requests if your log records or tests indicate that an application you are deploying causes significant problems on your server, or encounters significant problems, after the application handles a certain number of requests. Then, for the worker process assigned to the application pool containing the unhealthy application, set the number of requests to be handled before the worker process is recycled.

For information about how to configure a worker process to be recycled after processing a set number of requests through IIS Manager, see “Configure Application Pool Recycling” in “IIS Deployment Procedures” in this book. For information about how to configure a worker process to be recycled after processing a set number of requests by using the command-shell or scripts, see the **PeriodicRestartRequests** property in the “Metabase Property Reference” in IIS 6.0 Help.

Recycling at Scheduled Times

To avoid recycling worker processes when server demand is heaviest, you can configure an application pool to recycle the worker process, or worker processes, that are assigned to it at a specified time of day, which is based on a 24-hour format. By default, recycling at a scheduled time is disabled in IIS 6.0.

You might need to schedule some, or all, of your applications to recycle worker processes at times of the day when the server load is light. Enable recycling at a scheduled time so that worker processes are rescheduled at times that best fit the load. Be careful to stagger the recycling of your application pools because allowing them to recycle at the same time can degrade the performance of your server during the recycle period.

For information about how to use IIS Manager to configure a worker process to be recycled at scheduled times, see "Configure Application Pool Recycling" in "IIS Deployment Procedures" in this book. For information about how to configure a worker process to be recycled at scheduled times by using a command-line procedure or script, see the **PeriodicRestartSchedule** property in the "Metabase Property Reference" in IIS 6.0 Help.

Recycling on a Virtual-Memory Threshold

A problem application might periodically increase virtual memory without subsequently releasing it. In this way, virtual memory can increase indefinitely. To limit the impact of increased virtual memory, you can set an application pool to recycle the worker processes that are assigned to it whenever virtual memory reaches a specified threshold.

By default, recycling on a virtual-memory threshold is disabled in IIS 6.0. When this feature is enabled, the default virtual-memory setting is 500 megabytes (MB). Enable recycling on a virtual-memory threshold and set a virtual memory threshold that prevents unacceptable degradation. Begin with a setting of 70 percent of available virtual memory.

For information about using IIS Manager to configure a worker process to be recycled after consuming a set amount of memory, see "Configure Application Pool Recycling" in "IIS Deployment Procedures" in this book. For information about how to configure a worker process to be recycled after consuming a set amount of memory by using a command-line procedure or scripts, see the **PeriodicRestartMemory** property in the "Metabase Property Reference" in IIS 6.0 Help.

Recycling on a Used-Memory Threshold

A problem application might indefinitely increase the amount of memory that it uses without releasing the memory. This is also known as an application memory leak. You can set an application pool to recycle the worker processes that are assigned to it whenever used memory, which is also called private memory, reaches a specified threshold.

By default, recycling on a used-memory threshold is disabled in IIS 6.0. When it is turned on, the default used-memory threshold is 192 MB. To limit the impact from an application that increases used memory over time, enable recycling on a used-memory threshold and set a threshold that prevents unacceptable degradation such as a memory-bound application that is unable to process requests. Initially, try a threshold of 60 percent of available memory.

For information about using IIS Manager to configure a worker process to be recycled after consuming a set amount of memory, see "Configure Application Pool Recycling" in "IIS Deployment Procedures" in this book. For information about how to configure a worker process to be recycled after consuming a set amount of memory by using a command-line procedure or scripts, see the **PeriodicRestartPrivateMemory** property in the "Metabase Property Reference" in IIS 6.0 Help.

Tuning Performance

When IIS is running in worker process isolation mode, you can tune the performance of your applications to conserve CPU and memory resources, and to add worker processes to application pools that need them. By setting the application pool properties for idle time-out, you can optimize the time that worker processes remain idle for each application pool, removing them from memory. Another way to conserve memory is to shorten the request queue length limit property for the worker processes in each application pool.

You can increase the number of worker processes that are available to an application pool by configuring a *Web garden*, which is an application pool that is served by more than one worker process. When a worker process is engaged, another worker process can continue to process requests, enhancing server performance. Finally, on a computer with more than one CPU, you can establish affinity between worker processes and specific CPUs, ensuring that applications are tied to processors in accordance with your priorities.

Managing Isolation and Performance

To help achieve high levels of application availability, you can assign fewer applications to each application pool, as low as one application per application pool. This increases isolation, protecting applications from each other. However, configuring fewer applications per application pool increases the total number of application pools. This results in greater CPU and memory usage, degrading server performance, and resulting in reduced request-processing throughput.

The performance impact of increasing the number of application pools on a computer varies based on the number of application pools that are running worker processes. If there are a large number application pools, but few worker processes are running, the impact is relatively small; if there are a large number application pools and many worker processes are running, worker processes can impact each other as they use up memory and CPU capacity, thereby negatively impacting performance.

The degree to which increasing application isolation causes performance degradation can only be determined by testing. You can test for the optimum level of isolation by doing the following:

21. Set the highest level of isolation that your configuration can support, and test with all application pools running simultaneously. Assess the results after using Performance Monitor to record requests processed per second, CPU usage, and memory usage. If performance is satisfactory, the test is complete; if performance is not satisfactory, proceed to Step 2.
22. If the results in Step 1 are unsatisfactory, reduce the isolation level by reducing the number of application pools, and then run the test again.
23. Continue Step 2 until you achieve satisfactory performance. In addition to reducing the number of application pools, you might need to remove applications from the server.

For information about isolating applications, see "Isolate Applications in Worker Process Isolation Mode" in "IIS Deployment Procedures" in this book. For information about assigning applications to application pools by using a command-line procedure or scripts, see the **AppPoolId** property in the "Metabase Property Reference" in IIS 6.0 Help.

Configuring Idle Time-out for Worker Processes

To conserve system resources, configure the idle time-out interval of an application pool to shut down its worker process, or worker processes, after a specified period of idle time. This reduces memory consumption and CPU overhead. This also helps you manage the resources on a Web server when the processing load is heavy, when identified applications consistently fall into an idle state, or when new processing space is not available. To preserve performance, as the number application pools running simultaneously on a server increases, the idle timeout setting should be decreased.

By default, idle time-out is enabled, and the default time-out period is 20 minutes. To automatically shut down worker processes as they become idle, configure a scheduled shutdown of the idle worker process, or worker processes.

For information about how to configure idle time-out by using IIS Manager, see "Configure Application Pool Performance" in "IIS Deployment Procedures" in this book. For information about how to configure idle time-out by using a command-line procedure or scripts, see the **IdleTimeout** property in the "Metabase Property Reference" in IIS 6.0 Help.

Configuring a Request Queue Limit

When an application pool receives requests faster than it can handle them, the unprocessed requests might consume all of the memory on the server, slowing the server and preventing other application pools from processing requests. This can happen when the queue size limit for the requests is large and legitimate requests are coming in at a rapid rate, or during a denial of service attack.

To prevent requests from consuming all the memory for an application pool, limit the size of the request queue for the application pool. To monitor and manage CPU consumption, you must enable CPU monitoring, gather data on CPU usage, and reduce the request queue limit when CPU consumption exceeds the specified limit.

For example, if you have a Web server that is capable of processing 10,000 requests per minute, and you have an application that can queue up 5,000 requests within that period of time, the application consumes 50 percent of the capacity of the server. You can configure IIS 6.0 to limit the number of requests to 2,000, reducing the potential memory consumption of the application and reducing the Web server capacity used by the application to 20 percent.

By default, request queue limit is enabled in IIS 6.0, and the default value is 1,000 requests. When resetting the default value, remember that a value that is too high can prevent other applications from having an equal share of the server resources and a value that is too low can cause the server to return 503 errors, or a disruption of service for the clients.

For information about how to configure a request queue limit by using IIS Manager, see "Configure Application Pool Performance" in "IIS Deployment Procedures" in this book. For information about how to configure a request queue limit by using a command-line procedure or scripts, see the **AppPoolQueueLength** property in the "Metabase Property Reference" in IIS 6.0 Help.

Managing ASP.NET and Request Queue Limit

Recycling worker processes that are serving ASP.NET applications requires that you restart under a request processing load. Starting the new worker processes under a load might cause requests to be rejected because the queue is too full. When this occurs, either increase the queue size limit or disable recycling.

For more information about ASP.NET–specific considerations for applications, see "Deploying ASP.NET Applications in IIS 6.0" in this book.

Configuring Web Gardens

To improve availability for a Web site or application, you can increase the number of worker processes servicing the application pool by implementing a *Web garden*, an application pool that uses more than one worker process on a single computer. This might be helpful if you have already configured a single-application application pool and need to improve availability.

Creating a Web garden for an application pool enhances performance by providing the following benefits:

- **Robust processing of requests.** When a worker process in an application pool is tied up (for example, when a script engine stops responding), other worker processes can accept and process requests for the application pool.
- **Reduced contention for resources.** When a Web garden reaches a steady state, each new TCP/IP connection is assigned, according to a round-robin scheme, to a worker process in the Web garden. This helps smooth out workloads and reduce contention for resources that are bound to a worker process.

Because Web gardens enable the use of multiple processes, all processes have their own copy of application state, in-process session state, caches, and static data. Web gardens are not advantageous for applications that depend on application state. Identify the standard relative performance in a Web garden before you decide whether Web gardens are the best solution for your server.

Note

A Web garden differs from a *Web farm*, which distributes the load across multiple Web servers. You can design Web farms to provide your applications with the highest availability. For more information about Web farms, see "Scalability" in IIS 6.0 Help.

By default, the Web garden feature is disabled in IIS 6.0, and the default number of worker processes assigned to an application pool is set to one. You can enable Web garden functionality by setting the number of worker processes assigned to an application pool to a number greater than one.

For information about how to configure a Web garden by using IIS Manager, see "Configure Application Pool Performance" in "IIS Deployment Procedures" in this book. For information about how to configure a Web garden by using a command-line procedure or scripts, the **MaxProcesses** property in the "Metabase Property Reference" in IIS 6.0 Help.

Setting Processor Affinity on Servers with Multiple CPUs

On a multi-CPU server, you can configure application pools to establish affinity between worker processes and CPUs to take advantage of more frequent CPU cache hits. By using processor affinity, you can control processors when applications place demands on system resources. Processor affinity is used in conjunction with the processor affinity mask setting to specify the CPUs.

Use processor affinity on computers with multiple processors when it is necessary to continue processing an application that is tied to a subset of the hardware in such a way that transitions from one processor to another are minimized. By default, processor affinity is disabled in IIS 6.0, and the load is distributed across all available CPUs. On a server that is supporting multiple applications, you can enable processor affinity so that a processor is dedicated to each application.

For information about how to set processor affinity, see "Set Processor Affinity" in "IIS Deployment Procedures" in this book, or see the **SMPAffinitized**, and **SMPProcessorAffinityMask** properties in the "Metabase Property Reference" in IIS 6.0 Help.

Managing Application Pool Health

IIS 6.0 health-monitoring and management features support application availability by detecting problems with worker processes that are assigned to application pools, and by taking action to manage the problems. When IIS 6.0 is running in worker process isolation mode, you can detect the health condition of a specific application pool by enabling WWW service to ping a worker process that is assigned to it. Also, you can set features to take corrective action when an unhealthy condition occurs.

Generally, a healthy worker process responds to periodic pings that are sent by WWW service. If a worker process does not respond to the pings within the time interval specified by the **PingResponseTime** property, the WWW service determines the worker process to be unhealthy and triggers other health management features.

Consider using worker process health detection for the following reasons:

- It allows IIS to determine whether each process has at least one thread available to work on requests, and then decide whether to pull new requests from HTTP.sys or to continue processing existing requests.
- It allows the worker process to request recycles when Internet Server Application Programming Interface (ISAPI) extensions such as ASP.NET and ASP are identified to be unhealthy. The health detection feature uses the ability of a worker process to respond to a ping message. The worker process responds to a ping by requesting a recycle if one or more of the ISAPI extensions it has loaded is identified to be unhealthy.

Configuring Worker Process Pinging

Application pool health detection is the ability of the WWW service to detect that a worker process assigned to the application pool is in an unhealthy state by periodically *pinging*, or sending a ping message to, the worker process.

Optimizing Ping Intervals

You can specify a value for the ping interval by examining the potential amount of system resources that can be consumed by unhealthy worker processes over the interval (the default is 30 seconds). Then determine how long you can wait for system resources that unhealthy worker processes are using to be freed so that they are available to other worker processes.

If a worker process becomes unhealthy, the client does not receive a response until another worker process is started to replace the unhealthy worker process. This leaves the unhealthy worker process active without releasing the system resources used by the process. Over a period of time, the amount of system resources used by an unhealthy worker process can accumulate and affect the overall performance of the Web server.

For example, you might have an application that causes worker processes to become unstable. Over a five-minute period of time, the amount of system resources consumed by these unstable worker processes can consume Web server resources and affect the performance of other applications that are on the same Web server. You can configure IIS 6.0 to ping worker processes in that application pool to check the health of the worker processes every 30 seconds, which ensures that the unhealthy worker processes are shut down before system resources are consumed.

Set ping intervals to avoid false failure conditions caused by healthy worker processes that do not respond within the specified response time. You can check this by testing to ensure that the WWW service does not shut down a worker process that is still running, but is too busy to return the ping in the specified response time. If your tests indicate that your worker processes are shutting down because of false failure conditions, increase the ping response time, or alter the load on the worker process.

By default, worker process pinging is enabled in IIS 6.0. The default ping interval is 30 seconds, and the default ping response time is 60 seconds.

For information about how to configure worker process pinging by using IIS Manager, see "Configure Application Pool Health" in "IIS Deployment Procedures" in this book. For information about how to configure worker process pinging by using a command-line procedure or scripts, see the **PingingEnabled**, **PingInterval**, and **PingResponseTime** properties in the "Metabase Property Reference" IIS 6.0 Help.

Configuring Rapid-Fail Protection for Worker Processes

When WWW service detects that a worker process fails more than a set maximum number of times in a specified time period, it places the application pool that the worker process is serving in a rapid-fail state. Rapid-fail protection reduces processing overhead for unhealthy applications because the requests do not enter user-mode processing. As a result, applications that are running in other application pools are protected from the unhealthy application.

By default, rapid fail protection is enabled in IIS 6.0, with a maximum of 5 failures in a 10-minute interval. You can specify the threshold for placing an application pool in rapid-fail state by setting the maximum number of worker process failures that you want to allow over a specific period of time. Specify a threshold that allows IIS 6.0 to detect that a significant number of the worker processes in an application pool are failing and prevent system resources from being consumed.

For information about how to configure rapid-fail protection by using IIS Manager, see "Configure Application Pool Health" in "IIS Deployment Procedures" in this book. For information about how to configure rapid-fail protection by using a command-line procedure or scripts, see the **RapidFailProtection**, **RapidFailProtectionMaxCrashes**, and **RapidFailProtectionInterval** properties in the "Metabase Property Reference" in IIS 6.0 Help.

Configuring the Startup Time Limit for Worker Processes

The startup time limit specifies how long IIS will wait for a worker process to start servicing requests. By default, worker process startup time limit is enabled in IIS 6.0 with a default setting of 90 seconds. If 90 seconds is not long enough to allow the application to begin processing requests, then try a longer time interval.

For information about how to configure the startup time limit for worker processes by using IIS Manager, see "Configure Application Pool Health" in "IIS Deployment Procedures" in this book. For information about how to configure the startup time limit for worker processes by using a command-line procedure or scripts, see the **StartupTimeLimit** property in the "Metabase Property Reference" in IIS 6.0 Help.

Configuring the Shutdown Time Limit for Worker Processes

When IIS 6.0 detects that a worker process is unhealthy, IIS 6.0 marks the worker process for termination. The shutdown time limit specifies how long IIS waits until forcibly terminating the worker process when the worker process does not shut down automatically.

By default, worker process shutdown time limit is enabled in IIS 6.0 with a default of 90 seconds. If you have an accurate profile of requests, specify a shutdown time limit that is long enough to allow the worker process to complete any pending requests before shutting down.

For information about how to configure the shutdown time limit for worker processes by using IIS Manager, see "Configure Application Pool Health" in "IIS Deployment Procedures" in this book. For information about how to configure the shutdown time limit for worker processes by using a command-line procedure or scripts, see the **ShutdownTimeLimit** property in the "Metabase Property Reference" in IIS 6.0 Help.

Enabling Debugging for Application Pool Failures

Debugging a specific worker process that is serving an application pool helps you build high-availability applications. By default, debugging is disabled in IIS 6.0. When you enable debugging, the WWW service stops managing worker processes that fail to respond to a ping in the specified time period. Rather than terminating the application when a process fails to respond to a ping, IIS executes an action such as a process dump or an e-mail notification to alert you to the problem. You can then review the application to identify the cause of the failure.

Note

Enable debugging on an application pool only if the application pool continues to fail to respond to pings. The unhealthy process that is left running will continue to take up system resources until the process is terminated by the administrator or until IIS causes it to self-terminate. If you enable debugging, be sure to terminate the worker process after the debugging action is finished.

For more information about how to enable debugging, see "Debug Application Pool Failures" in "IIS Deployment Procedures" in this book, or see the **OrphanWorkerProcess**, **OrphanActionExe**, and **OrphanActionParams** properties in the "Metabase Property Reference" in IIS 6.0 Help.

Configuring Application Pool Identity

Application pool identity is the user account that serves as the process identity for the worker processes that are servicing the application pool. *Process identity* is the account that a process runs under. Every Windows Server 2003 process has a process identity that is used to control access to resources on the system.

Application pool identity can be assigned to a predefined account. Predefined accounts are known as *service-user accounts* and they are created by the operating system. You can use the predefined NetworkService, LocalSystem, or LocalService accounts for the application pool identity.

In IIS 6.0 worker process isolation mode, application pools have a default identity of NetworkService. The NetworkService identity has minimal administrative credentials, which helps reduce the attack surface of your Web server. If you deploy applications that require a fixed worker process identity, you can change the application pool identity. For information about security and application pool identities, see "Configuring Application Pool Identity Settings" in "Deploying ASP.NET Applications on IIS 6.0" in this book.

For information about how to configure application pool identity by using IIS Manager, see "Configure Application Pool Identity" in "IIS Deployment Procedures" in this book. For information about how to configure application pool identity by using a command-line procedure or scripts, see the **AppPoolIdentityType** property in the "Metabase Property Reference" in IIS 6.0 Help.

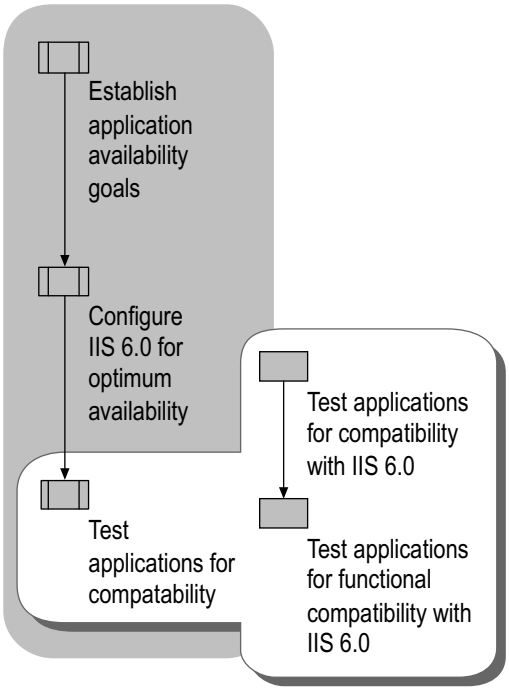
Testing Applications for Compatibility

Whether you are deploying existing applications that were developed for earlier versions of IIS on a server running IIS 6.0 in worker process isolation mode, or you are deploying new applications, it is important to test the applications for compatibility with worker process isolation mode features.

Functional testing consists of installing and configuring applications on a test server, and then sending HTTP requests to the application from a client on another computer. Load testing is functional testing that consists of sending HTTP requests in rapid succession to stress the worker process and the server. In both cases, you must diagnose the results for possible incompatibilities.

Figure 4.5 shows the process for testing application compatibility with worker process isolation mode.

Figure 4.5 Testing Applications for Compatibility



Checking for Known Conflicts with IIS 6.0

Most Web applications that run on earlier versions of IIS will run on IIS 6.0 in worker process isolation mode. However, in some cases, applications cannot run in worker process isolation mode. In other cases, you might be required to disable certain features to run an application in worker process isolation mode.

Applications that have the following characteristics are incompatible with worker process isolation mode, or must have certain features disabled to run in worker process isolation mode:

- **Require Read Raw Data Filters.** Applications that require Read Raw Data Filters are not compatible with worker process isolation mode. The Read Raw Data Filters feature is only supported in IIS 5.0 isolation mode.
- **Require Digest authentication.** You must use LocalSystem as the application pool identity for standard Digest authentication. This restriction does not apply when you are using Advanced Digest authentication.
- **Run only as a single instance.** When an application only runs as a single instance, you cannot configure Web gardens, you cannot run the application in more than one application pool, and you cannot use overlapped recycling.
- **URL length exceeds 16 kilobytes.** The default request header limit for HTTP.SYS is 16KB.
- **ASP applications that use consecutive dots in an include file.** Using two or more consecutive dots in an include file only works if parent paths are enabled. Alternatively, consider not using includes files.
- **Dependency on Inetinfo.exe.** The application requires Inetinfo.exe, but the Inetinfo.exe process does not run in worker process isolation mode. This occurs rarely, if at all.
- **Require Dllhost.exe.** The application requires Dllhost.exe, but the Dllhost.exe process is not available in worker process isolation mode. This occurs rarely, if at all.

Testing Applications for Compatibility with IIS 6.0

You can begin to test the compatibility of an application during the setup process for each application that you install on IIS 6.0.

To perform a setup test, complete the following steps:

24. Install the application by using the setup software provided.
25. Examine the IIS log (in \\Windows\System32\LogFiles\W3SVC*xxx*) for messages that indicate a WWW service failure. Also, look at the system Event log for 500-level errors.

Some common incompatibilities can be identified when you set up applications to run in IIS 6.0 worker process isolation mode. You can recognize these common incompatibilities by their symptoms, and then take the appropriate action to resolve them.

Version Detection Incompatibility

During setup, some applications fail to detect the IIS version correctly because they try to read non-standard registry keys that do not exist in IIS 6.0. The supported version information is under the following registry key: **HKLM\SOFTWARE\Microsoft\InetStp**

Symptoms

Some applications check for a 5 or a 4 in the version information, and then quit setup if the value is 6.

Non-standard registry locations where applications might look for version information include the following:

```
HKLM\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters

HKLM\SOFTWARE\Microsoft\InetMGR\Parameters
```

Resolution

Add the registry keys that the application is searching for to the registry. Otherwise, you cannot run the applications until you modify them to search for an existing registry key.

Wwwroot and Scripts Retrieval Incompatibility

Some applications use a nonexistent registry key to retrieve wwwroot and scripts. When this is true, the following message appears during setup:

```
HKLM/System/CurrentControlSet/W3svc/Parameters/Virtual Roots
"/ "=REZ_SZ:<physical location of wwwroot>
"/Scripts"=REZ_SZ:<physical location of scripts>
```

Symptom

Setup shows no default value for the wwwroot and scripts directories.

Resolution

Add the registry keys that the application is searching for to the registry. Otherwise, you cannot run the applications until you modify them to search for an existing registry key.

Testing Applications for Functional Compatibility with IIS 6.0

You can test the functional compatibility of an application running in worker process isolation mode by installing the application on a test server, and then sending HTTP requests to the application from another computer. You can identify the following functional incompatibilities when you send correctly composed HTTP requests to applications that were developed for earlier versions of IIS, or to new applications yet to be tested.

ISAPIs Not Enabled in the Web Service Extensions Node

Some application setups install script maps for their script engine but do not enable the script engine, and the documentation for the application might not mention the need to set the properties (by using IIS Manager, for example).

Symptoms

If your log file (%systemroot%\system32\logfiles\w3svc*Filename*.log) returns a 404 error with suberror code 2, then your application is trying to use a disabled ISAPI or Common Gateway Interface (CGI) file.

Resolution

Use the Web Service Extensions node in IIS Manager to enable the ISAPI or CGI file. Be sure to enter the exact path entry that is in the scriptmaps setting in the path part of the entry.

MIME Types Incompatibility

Your application might use a file name extension that is not recognized by IIS 6.0.

Symptoms

If your WWW service log file, which is located in %systemroot%\system32\logfiles\w3svc*Filename*.log, returns a 404 error with suberror code 3, then a file name extension is not recognized by IIS 6.0.

Resolution

Manually add the file name extension to the computer-level properties in IIS Manager. For more information about configuring MIME types, see "Configure MIME Types" in "IIS Deployment Procedures" in this book.

Recycling Incompatibility

The configurable recycling feature in IIS 6.0 might expose some design flaws in your applications. The most common problem is that applications take too long to start up or to shut down. For example, it is not a problem in IIS 5.0 if an application receives an access violation when shutting down. However, an IIS 6.0 administrator might see error messages in the Event log indicating that the application pool is taking too long to shut down. If the application needs to warm up when it starts up, then recycling causes the application to run at low efficiency every time it starts up. If the application does not save its state data when it shuts down, it loses state data often. Test the impact of the recycling period by configuring it to a shorter time and continuing to check the log.

Symptoms

You can identify recycling incompatibility problems by testing the worker process with a short recycle period. If problems exist in the applications in the application pool that is served by the worker process, the following symptoms might occur:

- If an application is shutting down slowly, you might find error messages in the Event log, which indicate that the worker process is exceeding its shutdown time limit.
- If the application needs time to warm up when it starts, response times might be very slow at first.

Resolution

The application might require extensive modification. While running the application unmodified, you cannot use recycling. For more information about configuring the features for worker process recycling, see "Configure Application Pool Recycling" in "IIS Deployment Procedures" in this book.

Overlapped Recycling and Single Instance Application Incompatibility

If your application cannot run as multiple instances, then overlapped recycling fails. The failure occurs because the new worker process is running before the existing worker process is terminated, resulting in an attempt to run two instances of the same application.

Symptoms

The symptoms are synchronization-related, such as an access denied error that appears in the IIS log (in \\Windows\System32\LogFiles\W3SVC*xxx*).

Resolution

Do not use overlapped recycling. For more information about configuring the features for recycling worker processes, see "Configure Application Pool Recycling" in "IIS Deployment Procedures" in this book.

Subauthentication Is Not Installed

Subauthentication is not installed by default. You require subauthentication if both of the following are true:

- You are using Digest authentication on your server running IIS 6.0.
- The domain controller is running IIS 5.0.

Symptom

Digest authentication fails.

Resolution

Install subauthentication on your server. For information about how to install subauthentication, see "Install Subauthentication" in "IIS Deployment Procedures" in this book.

Additional Resources

These resources contain additional information and tools related to this chapter.

Related Information

- "Deploying ASP.NET Applications in IIS 6.0" in this book for more information about ASP.NET–specific considerations for applications.
- "Securing Web Sites and Applications" in this book for information about securing IIS applications.

- "Migrating IIS Web Sites to IIS 6.0" in this book for information about migrating IIS applications.
- "Migrating Apache Web Sites to IIS 6.0" in this book for information about migrating Apache Web sites.
- "IIS Deployment Procedures" in this book for more information about procedures for ensuring application availability.
- The MSDN Online link on the Web Resources page at http://www.microsoft.com/windows/reskits/webresources for more information about the HSE_REQ_REPORT_UNHEALTHY support function. Search for ISAPI Reference, and select Extension Reference.

Related Help Topics

- The "Metabase Property Reference" in IIS 6.0 Help, which is accessible from IIS Manager, for more information about programmatically configuring IIS for optimum availability.
- "Scalability" in IIS 6.0 Help, which is accessible from IIS Manager, for more information about Web farms.